11-1-2014

Designing a Bayer Filter with Smooth Hue Transition Interpolation Using the Xilinx System Generator

Zhiqiang Li
University of Nebraska-Lincoln, zli13@unl.edu

Peter Revesz
University of Nebraska-Lincoln, prevesz1@unl.edu

Designing a Bayer Filter with Smooth Hue Transition Interpolation Using the Xilinx System Generator

Zhiqiang Li, Peter Z. Revesz

***Abstract*—**This paper describes the design of a Bayer filter with smooth hue transition using the System Generator for DSP. We describe and compare experimentally two different designs, one based on a MATLAB implementation and the other based on a modification of the Bayer filter using bilinear interpolation.

***Keywords*—**Bayer array, Demosaicing, FPGA, Interpolation.

I. Introduction

Digital cameras perform a sequence of complicated processing steps while recording color images. A color image usually contains three different color components in each pixel: red (R), green (G) and blue (B). Digital cameras use three separate sensors to capture these three components [1]. In order to reduce the cost, digital cameras capture images using a sensor overlaid with a color filter array (CFA). CFAs allow only one color component for each pixel, which means we need to generate the full color images from the output of the image sensor [2].

Bayer color filter arrays (Bayer CFAs) are currently one of the most common CFAs in digital cameras and can be used together with many different interpolation methods [3, 4]. The *System Generator for DSP,* commonly referred to as just *System Generator* [5, 6], is a MATLAB/Simulink-based simulation tool from Xilinx Inc. [7]. The System Generator is a hardware design package that allows programming on the FPGA and modeling a system using Simulink. The System Generator contains many modules, such as FIR filter, FFT, FIFO, RAM and ROM.

II. The Bayer Color Filter Array

Bayer CFAs greatly reduce the complexity and the cost of digital cameras. Each Bayer CFA contains twice as many green elements than red or blue ones, reflecting the fact that the cone cells in the human retina are most sensitive to green light. The full color image contains of three components (R, G and B) in each pixel, but a Bayer image, which is the output of a Bayer CFA, contains only one component in each pixel. However, from a Bayer image a full color image is generated by *demosaicing* [8], that is, an interpolation that estimates the values of the missing components [9]. For *demosaicing*, Xilinx uses *bilinear* interpolation, which performs the following three steps:

1. Estimate the missing green values in the red and blue pixels by using their four green neighbors. For example, using the Bayer image in Fig 1., the bilinear interpolation finds:

$$G8 = (G3 + G7 + G9 + G13)/4$$
$$G14 = (G9 + G19 + G13 + G15)/4 \quad (1)$$

2. Estimate the missing red or blue values in the green pixels:

$$B7 = (B6 + B8)/2$$
$$R7 = (R2 + R12)/2 \quad (2)$$

3. Estimate the missing red value of the blue pixel and the missing blue values of the red pixels:

$$R8 = (R2 + R4 + R12 + R14)/4$$
$$B12 = (B6 + B8 + B16 + B18)/4 \quad (3)$$

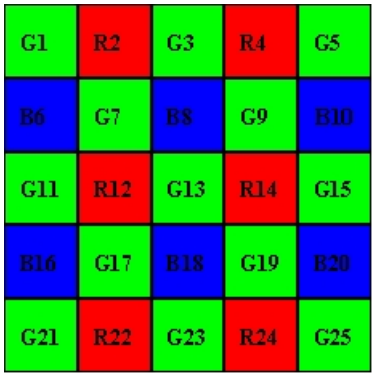

Fig. 1 A Bayer color filter array

Instead of the *bilinear* interpolation, this paper uses *smooth hue transition* interpolation [10]. Before estimating the missing red and blue values, we first estimate the missing green values of the red or blue pixels, the same way as in step (1) of the bilinear interpolation. Let the blue hue be B/G and the red hue R/G. These are used the estimate the missing blue

Zhiqiang Li is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588, USA (zli@cse.unl.edu).

Peter Z. Revesz is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588, USA (revesz@cse.unl.edu).

ISBN: 978-1-61804-256-9

element of the green pixels by:

$$B7 = (G7/2) \times (B6/G6 + B8/G8)$$
$$B13 = (G13/2)(B8/G8 + B18/G18) \quad (4)$$

Then the missing red values of the green pixels are estimated by:

$$R13 = (G13/2) \times (R12/G12 + R14/G14)$$
$$R7 = (G7/2) \times (R2/G2 + R12/G12) \quad (5)$$

The missing red values of the blue pixels are estimated by:

$$R8 = (G8/4) \times (R2/G2 + R4/G4 + R12/G12 + R14/G14) \quad (6)$$

Finally, the missing blue values of the red pixels are estimated by:

$$B12 = (G12/4) \times (B6/G6 + B8/G8 + B16/G16 + B18/G18) \quad (7)$$

III. Development Platforms

Xilinx is a supplier of programmable logic devices. It is famous for inventing the field programmable gate array (FPGA). Xilinx Spartan®-3A DSP [11] FPGA video starter kit (VSK) is a development platform consisting of the Spartan-3A DSP 3400A development platform, the FMC-video daughter card and a VGA camera. This platform enables experimenting with video processing using the Spartan-3A DSP family of FPGAs. VSK also includes a variety of software components, which are the Xilinx ISE® Design Suite 11.1 (includes as well as full versions of EDK and System Generator).

System Generator is a design tool that enables us to use the Mathworks model-based design environment Simulink for FPGA design. Developers do not need to have experience with FPGAs or RTL design when using System Generator. The Simulink modeling environment with a Xilinx specific blockset is used to complete the design. The downstream FPGA implementation steps are automatically performed to generate an FPGA programming file. Over 90 DSP blocks are provided in the Xilinx DSP blockset for Simulink. Common blocks such as adders, multipliers and registers are included. In addition, some complex building blocks, such as FFTs, filters and memories are also provided. The System Generator is based on Simulink from MATLAB.

IV. Implementation in MATLAB of the Smooth Hue Transition Interpolation

We use MATLAB to implement the smooth hue transition interpolation. First, the Bayer image is captured using DH-SV1410, which is widely used in industry. The following algorithm assumes that the size of the input data is a 1040 by 1392 Bayer image. We apply the smooth hue transition interpolation algorithm to the Bayer image to reconstruct the full color image. The function

result_g = shtlin_g_rg (Bayer)

implements Step (1), where result_g stands for the green values. The function

result_r = shtlin_r_rg (Bayer, result_g)

implements Steps (5) and (6) where result_r stands for the red values. The function

result_b = shtlin_b_rg (Bayer, result_g)

implements Steps (4) and (7) where result_b stands for the blue elements. The red, green and blue components constitute the interpolated 1040 by 1392 full color image. Fig. 2 shows an example of a smooth hue transition interpolation.

Fig. 2 Smooth hue transition by MATLAB

V. Reference Design from Xilinx

A. Overall design of the camera

Xilinx provides a reference Bayer filter design using bilinear interpolation, which can be described as follows. One big block called "vsk_camera_vop" is actually the design of a camera. There are three inputs and six outputs. "vsync", "hsync" are the vertical and horizontal synchronization signal of the oscilloscope. "BayerRaw_Raw" contains the one-dimensional Bayer image. "vs_out", "hs_out" are output synchronization signals. "red_out", "green_out" and "blue_out" stand for the color output information. "de_out" is the output enable signal. All of the output signals are connected to the oscilloscope block. The camera pipeline consists of several functional blocks:

- "dyn_range_exp", the dynamic range expansion [12] module, in photography, dynamic range describes the ratio between the maximum and the minimum light intensities. The higher the dynamic range is, the better the image is.
- "spc", the stuck pixel correction [13] module. Some of the pixels cannot be displayed correctly,

we need to correct this.

- "bright_contrast", we also need to control the brightness and contrast of the image.
- "bayer_filter", this is the module we need to focus on, using the bilinear interpolation to complete the reconstruction.
- "color_balance", this module is used to adjust the overall intensity of pixels, making the image look better.
- "stats", this module is used to calculate the maximum and minimum value of the pixels.

B. Bayer filter design

Since the details of the Bayer filter design (see Fig. 3) are complicated, we explain here only the bilinear algorithm using as an example Fig. 1. The block "Delay9" makes sure that the data is synchronized with the vertical and horizontal signals. Numbers from 1 to 9 stand for the location of pixels in the circuit. When the data reach location A, because of the block "delay 7", the data cannot reach location 1 until two clocks later. At the same time, data can reach the block "Single Port RAM", and is written into the RAM according to the address provided. The "Single Port RAM" is set to the mode "Read before write", which means one clock later, the data at location B is the initial value 0, not the data value stored. During the next clock, the address is incremented by 1, and new data is stored into the "Single Port RAM". Since the address is added at this time, the data at location B is still the initial value 0 at the address. From location 4, we can get the value 0 from the last clock. According to the description above, before the pixels in the first row (G1, R2, G3, R4, and G5) reach location A, values from location 4, 5, 6, 7, 8, 9 are all 0.

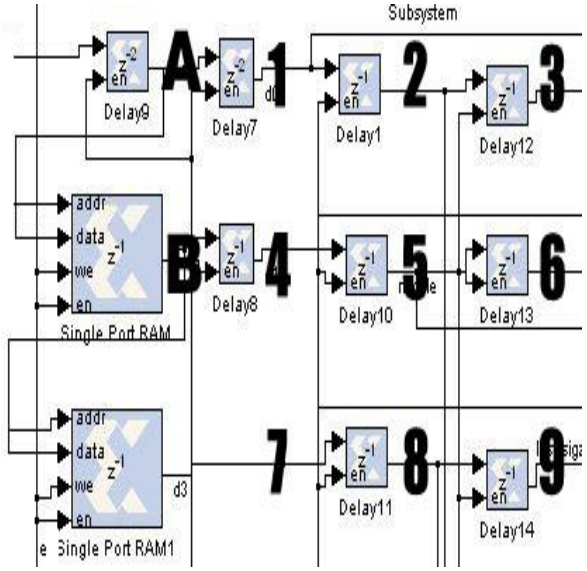

Fig. 3 Part of the design by bilinear

When the pixels in the second row (B6, G7, B8, G9, and B10) start to reach location A, because of the control from horizontal synchronization signal, the address is reset to 0. Similarly to the previous description, "Single Port RAM" is in the "Read before write" mode, the data we get from it is not the data at location A (the data of the second row), it should be the data from the first row (the data is G1). After one clock delay, the G1 appears at location B. Similarly, after getting the initial value 0 from "Single Port RAM1", G1 is stored into it. After all the pixels of the second row (B6, G7, B8, G9, B10) reach location A, the pixels of the first row (G1, R2, G3, R4, G5) are stored into "Single Port RAM1".

When the third row (G11, R12, G13, R14, G15) arrives to A, we can get the pixels of first row from "Single Port RAM1" and store the second row to it. We get the pixels of the second row from "Single Port RAM", and store the third row to it. Finally, locations 1, 2, 3 store the pixels from the third row, locations 4, 5, 6 store the pixels from the second row, and locations 7, 8, 9 store the pixels from the first row.

All of the delay blocks help make the pixels stay in the circuit temporarily, in order to apply the interpolation method to the pixels. For example, at one moment, G1, R2, G3, B6, G7, B8, G11, R12 and G13 can be obtained from location 1 to location 9. Then G7 is the center of the 3 by 3 array (G1, R2, G3, B6, G7, B8, G11, R12, and G13). Now G7 does not have any red and blue values but only has a green value. We directly connect location 5 to "Shift 2", and get the green value through "Mux2". In order to get the blue element, add the values from location 4 and location 6, and get it through "Mux3". The red element is similar, add the values from location 2 and location 8, and get it through "Mux4".

In the design, we get the average value via the slice block. Slice block is used to truncate the binary bits. For example, binary 1110 (decimal 14), and we remove the last bit 0, the result is 111 (decimal 7), which is 14 divided by 2. In this way, we can simplify some of the calculations.

VI. Bayer filter using Smooth hue transition interpolation

Modifying the Xilinx reference design, we designed a Bayer filter that uses a smooth hue transition interpolation. In the reference design, 9 locations (from location 1 to location 9) are needed to store the pixels temporarily, which actually is a 3 by 3 array. Our modified design uses a 5 by 5 array, that is, 25 locations (from location 1 to location 25). Fig. 4 shows part of the design. We again use Fig. 1 to explain the process.

At one moment, all of the pixels in Fig. 1 are corresponding to the locations in Fig. 4. For example, pixel G1 is at location 1, and pixel G13 is at location 13, etc. Further, G13 is the center of the array because it is a green element. We can directly forward its value to the Mux block. We use Step (1) to estimate G12 and G14 and the following to estimate the red values of element 13:

$$R13 = (G13/2) * (R12/G12 + R14/G14) \quad (8)$$

Besides, we need two additional blocks here, Multiplier and Divider. We add values at locations 11, 13, 7, 17, divide the

sum by 4, and get the green value G12 at location 12. We can also estimate G14 through the values at locations 13, 15, 9, 19. Notice that the values at location 12, 14 are red elements. We connect location 12 with G12, location 14 with G14, and calculate the quotients R12/G12 and R14/G14. Then we sum the two quotients by an adder and connect the sum and G13 with a multiplier. Finally, we pass the product to the shift block, right shift 1 bit (divide by 2) and get the red value R14. The estimation of the blue value at location 13 can be done similarly to the estimation of the red value.

At another moment, pixel R14 is the center of the array. Since there is a red value at location 14, we forward it to the "Mux" block. We get the green value using as in the bilinear interpolation. We estimate the blue value of this pixel using Step (9). The elements G8, G10, G18, G20 can be estimated via bilinear interpolation. Then in order to estimate the blue value at location 14, we calculate the quotients B8/G8, B10/G10, B18/G18 and B20/G20, sum the four quotients, multiply the sum by G14, and block shift the results, that is, to divide by four. That can be expressed using the formula:

$$B14 = G14 / 4 \times (B8 / G8 + B10 / G10 + B18 / G18 + B20 / G20) \quad (9)$$

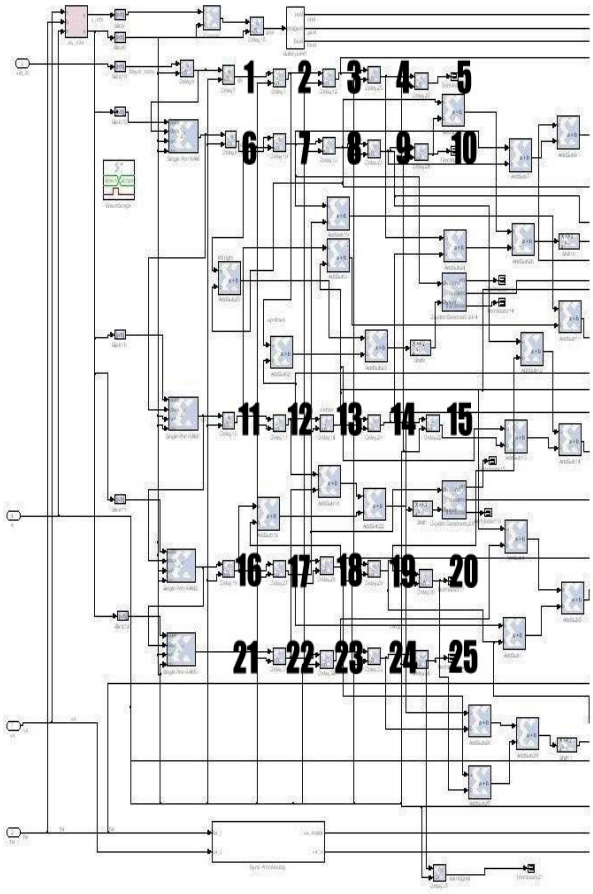

Fig. 4 Design by smooth hue transition

VII. Experiment Results

We use signal-noise ratios to compare the quality of the images reconstructed by our MATLAB implementation described in Section IV and our modified Bayer filter described in Section VI. As an example, Figs. 2 and 5 show a picture of the first author as reconstructed by these two methods, respectively. In addition, Table 1 shows the comparison result for the same image and another image, which is available from the first author's B.S. thesis.

Fig. 5 Smooth hue transition by System Generator

Table. 1 Signal-to-Noise Ratios

	MATLAB Implementation	Modified Bayer
fountain	4.7923	8.1590
person	9.5556	9.6273

The table suggests that modified Bayer filter is generally a better an interpolation method.

References

[1] *Image Sensor Architectures for Digital Cinematography,* DALSA Corp.

[2] X. Lia, B. Gunturkb and L. Zhangc, "Image Demosaicing: A Systematic Survey," *SPIE Proceedings,* vol. 6822, Jan. 2008.

[3] B. Bayer "Color imaging array," U.S. Patent 3 971 065, July 20, 1976.

[4] T. Wittman, "Mathematical Techniques for Image Interpolation," unpublished.

[5] *System Generator for DSP Getting Started Guide,* Xilinx Inc., 2012.

[6] *System Generator for DSP Reference Guide*, Xilinx Inc, Apr. 2008.

[7] Xilinx, Available: http://www.xilinx.com

[8] R. Kimmel, "Demosaicing: Image Reconstruction from Color CCD Samples," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 8, Sept. 1999, pp. 1221-1228.

[9] R. Lukac, "Color Filter Arrays: Design and Performance Analysis," *IEEE Transactions on Consumer Electronics,* vol. 51, No. 4, Nov. 2005, pp. 1260-1267.

[10] R. Maschal, S. Young, J. Reynolds, K. Krapels, J. Fanning, and T. Corbin, "Review of bayer pattern color filter array (cfa) demosaicing with new quality assessment algorithms," U.S. Army Res. Lab, 2010.

[11] *Spartan 3A-DSP FPGA Video Starter Kit User guide,* Xilinx Inc., Apr. 2008.

[12] S. Battiato, A. Castorina and M. Mancuso, "High dynamic range imaging for digital still camera: an overview," *Journal of Electronic Imaging,* vol. 12(3), Jul. 2003, pp. 459–469.

[13] A. A. Tanbakuchi, A. V. D. Sijde, B. Dillen, A. J. P. Theuwissen and W. d. Haan, "Adaptive pixel defect correction," *SPIE Proceedings,* vol. 5017, May. 2003.